3.23 Verification Techniques for Graph Rewriting (Tutorial)

Arend Rensink (University of Twente, NL)

This tutorial paints a high-level picture of the concepts involved in verification of graph transformation systems. We distinguish three fundamentally different application scenarios for graph rewriting: (1) as grammars (in which case we are interested in the language, or set, of terminal graphs for a fixed start graph); (2) as production systems (in which case we are interested in the relation between start and terminal graphs); or (3) as behavioural specifications (in which case we are interested in the transition system as a whole). We then list some types of questions one might want to answer through verification: confluence and termination, reachability, temporal properties, or contractual properties. Finally, we list some techniques that can help in providing answers: model checking, unfolding, assertional reasoning, and abstraction.

3.24 Refining Orderings for Parameterized Verification

Ahmed Rezine (Linköping University, SE)

Joint work of Ahmed Rezine; Zeinab Ganjei; Yu-Fang Chen; Parosh Abdulla; Giorgio Delzanno; Petru Elese; Zebo Peng
Main reference Z. Ganjei, A. Rezine, P. Eles, Z. Peng, "Abstracting and counting synchronizing processes," in Proc. of the 16th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'15), LNCS, Vol. 8931, pp. 227–244, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-662-46081-8_13
Main reference Z. Ganjei, A. Rezine, P. Eles, Z. Peng, "Lazy Constrained Monotonic Abstraction," in Proc. of the 17th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'16), LNCS, Vol. 9583, pp. 147–165, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-662-49122-5_7

Multi-threaded programs may synchronise in subtle ways. For instance, they can use integer variables to count the number of threads satisfying some property in order to implement dynamic barriers or to organise their interleaved execution. We address the problem of automatically establishing deadlock freedom and safety in general for multi-threaded programs generating an arbitrary number of concurrent processes. For this purpose, we explain how we leverage on simple techniques to derive "counting invariants", i.e., invariants that relate the number or processes in a given location to the values of the program variables. We use these invariants and leverage on predicate abstraction techniques in order to generate non-monotonic counter machine reachability problems that faithfully capture the correctness of the safety property.

We describe how we check reachability for non-monotonic counter machines. The idea is to localise the refinement of well quasi orderings in order to allow for a decidable reachability analysis on possibly infinite abstractions that are well structured wrt. these orderings. The orderings can be refined based on obtained false positives in a CEGAR like fashion. This allows for the verification of systems that are not monotonic and are hence inherently beyond the reach of classical well-structured-systems-based analysis techniques. Unlike classical lazy predicate abstraction, we show the feasibility of the approach even for systems with infinite control. Our heuristics are applicable both in backward and in forward as shown by our experiments.